CSE 373

Sorting 1: Bogo Sort, Stooge Sort, Bubble Sort
reading: Weiss Ch. 7

slides created by Marty Stepp
http://www.cs.washington.edu/373/

© University of Washington, all rights reserved.
Sorting

• **sorting**: Rearranging the values in an array or collection into a specific order (usually into their "natural ordering").
  
  - one of the fundamental problems in computer science
  - can be solved in many ways:
    - there are many sorting algorithms
    - some are faster/slower than others
    - some use more/less memory than others
    - some work better with specific kinds of data
    - some can utilize multiple computers / processors, ...

• **comparison-based sorting**: determining order by comparing pairs of elements:
  - `<`, `>`, `compareTo`, ...
Sorting methods in Java

• The Arrays and Collections classes in java.util have a static method sort that sorts the elements of an array/list

```java
String[] words = {"foo", "bar", "baz", "ball"};
Arrays.sort(words);
System.out.println(Arrays.toString(words));
// [ball, bar, baz, foo]

List<String> words2 = new ArrayList<String>();
for (String word : words) {
    words2.add(word);
}
Collections.sort(words2);
System.out.println(words2);
// [ball, bar, baz, foo]
```
## Collections class

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>binarySearch(list, value)</td>
<td>returns the index of the given value in a sorted list (&lt; 0 if not found)</td>
</tr>
<tr>
<td>copy(listTo, listFrom)</td>
<td>copies listFrom's elements to listTo</td>
</tr>
<tr>
<td>emptyList(), emptyMap(), emptySet()</td>
<td>returns a read-only collection of the given type that has no elements</td>
</tr>
<tr>
<td>fill(list, value)</td>
<td>sets every element in the list to have the given value</td>
</tr>
<tr>
<td>max(collection), min(collection)</td>
<td>returns largest/smallest element</td>
</tr>
<tr>
<td>replaceAll(list, old, new)</td>
<td>replaces an element value with another</td>
</tr>
<tr>
<td>reverse(list)</td>
<td>reverses the order of a list's elements</td>
</tr>
<tr>
<td>shuffle(list)</td>
<td>arranges elements into a random order</td>
</tr>
<tr>
<td>sort(list)</td>
<td>arranges elements into ascending order</td>
</tr>
</tbody>
</table>
Sorting algorithms

- **bogo sort**: shuffle and pray
- **bubble sort**: swap adjacent pairs that are out of order
- **selection sort**: look for the smallest element, move to front
- **insertion sort**: build an increasingly large sorted front portion
- **merge sort**: recursively divide the array in half and sort it
- **heap sort**: place the values into a sorted tree structure
- **quick sort**: recursively partition array based on a middle value

Other specialized sorting algorithms:

- **bucket sort**: cluster elements into smaller groups, sort them
- **radix sort**: sort integers by last digit, then 2nd to last, then ...
- ...
Bogo sort

- **bogo sort**: Orders a list of values by repetitively shuffling them and checking if they are sorted.
  - name comes from the word "bogus"; a.k.a. "bogus sort"

  The algorithm:
  - Scan the list, seeing if it is sorted. If so, stop.
  - Else, shuffle the values in the list and repeat.

- This sorting algorithm (obviously) has terrible performance!
  - What is its runtime?
Bogo sort code

// Places the elements of a into sorted order.
public static void bogoSort(int[] a) {
    while (!isSorted(a)) {
        shuffle(a);
    }
}

// Returns true if a's elements are in sorted order.
public static boolean isSorted(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        if (a[i] > a[i + 1]) {
            return false;
        }
    }
    return true;
}
// Shuffles an array of ints by randomly swapping each element with an element ahead of it in the array.
public static void shuffle(int[] a) {
    for (int i = 0; i < a.length - 1; i++) {
        // pick a random index in [i+1, a.length-1]
        int range = a.length - 1 - (i + 1) + 1;
        int j = (int) (Math.random() * range + (i + 1));
        swap(a, i, j);
    }
}

// Swaps a[i] with a[j].
public static final void swap(int[] a, int i, int j) {
    if (i != j) {
        int temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
• How long should we expect bogo sort to take?
  - related to probability of shuffling into sorted order
  - assuming shuffling code is fair, probability equals 1 / (number of permutations of $N$ elements)

$$P_N^N = N!$$

- average case performance: $O(N \times N!)$
- worst case performance: $O(\infty)$
- What is the best case performance?
Stooge sort

- **stooge sort**: A silly sorting algorithm with the following algorithm:

  `stoogeSort(a, min, max):`
  - if `a[min]` and `a[max]` are out of order: swap them.
  - stooge sort the first 2/3 of `a`.
  - stooge sort the last 2/3 of `a`.
  - stooge sort the first 2/3 of `a`, again.

  - Surprisingly, it works!
  - It is very inefficient. $O(N^{2.71})$ on average, slower than other sorts.
  - Named for the Three Stooges, where Moe would repeatedly slap the other two stooges, much like stooge sort repeatedly sorts 2/3 of the array multiple times.
Stooge sort example

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>9</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>call #1</td>
<td>5</td>
<td>6</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>#2</td>
<td>4</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#4</td>
<td>2</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#5</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#6</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#7</td>
<td>4</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#8</td>
<td>4</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#9</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#10</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#11-14</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#15</td>
<td>5</td>
<td>6</td>
<td>1</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#16</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#17</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#18</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#19</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#20-23</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#24-27</td>
<td>1</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#28</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>#29</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#30</td>
<td>1</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#31</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#32</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#33-36</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#37-40</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A total of 40 recursive calls are made! Ouch.

... calls 12-14 omitted (no swaps made)

... calls 21-23 omitted (no swaps made)

... calls 25-27 omitted (no swaps made)

... calls 34-36 omitted (no swaps made)

... calls 38-40 omitted (no swaps made)
Stooge sort code

```java
public static void stoogeSort(int[] a) {
    stoogeSort(a, 0, a.length - 1);
}

private static void stoogeSort(int[] a, int min, int max) {
    if (min < max) {
        if (a[min] > a[max]) {
            swap(a, min, max);
        }
        int oneThird = (max - min + 1) / 3;
        if (oneThird >= 1) {
            stoogeSort(a, min, max - oneThird);
            stoogeSort(a, min + oneThird, max);
            stoogeSort(a, min, max - oneThird);
        }
    }
}
```
Bubble sort

- **bubble sort**: orders a list of values by repetitively comparing neighboring elements and swapping their positions if necessary

- more specifically:
  - scan the entire list, exchanging adjacent elements if they are not in relative order; this bubbles the highest value to the top
  - scan the entire list again, bubbling up the second highest value
  - ... 
  - repeat until all elements have been placed in their proper order
"Bubbling" largest element

- Traverse a collection of elements
  - Move from the front to the end
  - "Bubble" largest value to end using pair comparisons and swapping
  - What can you assume about the array's state afterward?

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>42</td>
<td>77</td>
<td>35</td>
<td>12</td>
<td>91</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>42</td>
<td>35</td>
<td>12</td>
<td>77</td>
<td>8</td>
<td>91</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>35</td>
<td>42</td>
<td>12</td>
<td>42</td>
<td>8</td>
<td>91</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>35</td>
<td>12</td>
<td>42</td>
<td>8</td>
<td>77</td>
<td>91</td>
</tr>
</tbody>
</table>
// Places the elements of a into sorted order.
public static void bubbleSort(int[] a) {
    for (int i = 0; i < a.length; i++) {
        for (int j = 1; j < a.length - i; j++) {
            // swap adjacent out-of-order elements
            if (a[j - 1] > a[j]) {
                swap(a, j-1, j);
            }
        }
    }
}
An optimization

// Places the elements of a into sorted order.
public static void bubbleSort(int[] a) {
    for (int i = 0; i < a.length; i++) {
        boolean changed = false;
        for (int j = 1; j < a.length - i; j++) {
            // swap adjacent out-of-order elements
            if (a[j - 1] > a[j]) {
                swap(a, j-1, j);
                changed = true;
            }
        }
        // if j-loop does not make any swaps,
        // the array is now sorted, so stop looping
        if (!changed) {
            break;
        }
    }
}
Bubble sort runtime

• Running time (# comparisons) for input size $N$:

\[
\sum_{i=0}^{N-1} \sum_{j=1}^{N-i} 1 = \sum_{i=0}^{N-1} (N - i) \\
= N \sum_{i=0}^{N-1} 1 - \sum_{i=0}^{N-1} i \\
= N^2 - \frac{(N - 1)N}{2} \\
= O(N^2)
\]

- number of actual swaps performed depends on the data; out-of-order data performs many swaps
- runs slower the more elements are out-of-order; slowest on descending input, fastest on ascending (already-sorted) input
  - (the optimized version on previous slide is $O(N)$ for ascending input)